VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VYUŽITÍ LINUXOVÝCH KONTEJNERŮ PRO STÁLOST TESTŮ

USING OF LINUX CONTAINERS IN TEST FIXTURE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARIÁN ORSZÁGH**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19240/2016/xorsza00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Országh Marián**
Obor: Informační technologie
Téma: **Využití Linuxových kontejnerů pro stálost testů**
Using of Linux Containers in Test Fixture
Kategorie: Analýza a testování softwaru

Pokyny:

1. Nastudujte technologii Linuxových kontejnerů. Nastudujte projekt Docker.
2. Analyzujte požadavky pro stálost testů (test fixture) linuxových aplikací. Navrhněte program nebo sadu programů pro vytvoření test fixture testované aplikace pro jednoduchou reprodukci chyby. Výsledkem test fixture by měl být kontejner se všemi potřebnými závislostmi.
3. Implementujte program pro tvorbu test fixture pomocí projektu Docker.
4. Ověřte funkcionalitu programu na několika jednoduchých příkladech.

Literatura:

- Rosen, R.: Linux Containers and the Future Cloud. Linux Journal. 2014-06-10. URL: http://www.linuxjournal.com/content/linux-containers-and-future-cloud
- Domovská stránka projektu Docker. URL: https://www.docker.com/
- P. Ammann, J. Offutt. *Introduction to Software Testing*, Cambridge University Press, 2008. ISBN 978-0-511-39330-3.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá štúdiom Linuxových kontajnerov a ich aplikáciou na vytvorenie stabilného prostredia pre testovanie softvéru. Programové riešenie problém delí na niekoľko častí. Najprv sa podľa požiadavok používateľa vytvorí konfigurácia, následne sa podľa nej vytvorí kontajner a nakoniec sa tento kontajner spustí spolu s dodanými testami za použitia platformy Docker. Program sám o sebe naplňuje počiatočné požiadavky, avšak jeho funkcionalita nie je zaručená vzhľadom na využitie softvéru tretej strany ako napríklad správcovia balíkov, čo môže spôsobiť neočakávané chyby za behu programu. Hlavným prínosom práce je zaobalenie platformy Docker tak, aby od užívateľa vyžadovala minimálnu, alebo žiadnu znalosť platformy Docker a umožňovala vytvárať kontajnery zjednodušenou formou.

Abstract

The main focus of this thesis is the study of Linux containers and their application in creation of software test fixtures. The program solution divides the problem into several segments. At first, a configuration is set up in accordance with the user's specification, next a container is created according to given configuration, and in the end, the created container is launched alongside supplied tests while using the Docker platform. The program itself meets initial requirements although its functionality is not guaranteed as a result of usage of third-party software such as package managers, which may cause unexpected runtime errors. Primary asset of the thesis is the wrapping of the Docker platform to the degree, that its user needs minimal, or no knowledge of the platform, and allows them to create containers in a simplified way.

Kľúčové slová

Linuxové kontajnery, Docker, testovanie softvéru, stále prostredia pre testovanie

Keywords

Linux containers, Docker, software testing, test fixtures

Citácia

ORSZÁGH, Marián. *Využití Linuxových kontejnerů pro stálost testů.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrčka Aleš.

Využití Linuxových kontejnerů pro stálost testů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením doktora Aleša Smrčky. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Marián Országh
16. mája 2017

Poďakovanie

Týmto by som chcel poďakovať svojemu vedúcemu bakalárskej práce doktorovi Aleši Smrčkovi za podporu, odborné vedenie a pomoc pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce bolo naštudovať platformu Docker a v praktickej časti ju aplikovať do programu, ktorý by zapúzdril komunikáciu s virtualizačnou platformou do jednoducho ovládateľného programu, ktorý je určený špecificky na nazadzovanie testov do stabilných prostredí pre testovanie (angl. test fixtures - vzhľadom na to, že slovenský jazyk neobsahuje priamy preklad pre anglický termín test fixtures, pre účely tejto práce bol použítý termín stabilné prostredia pre testovanie).

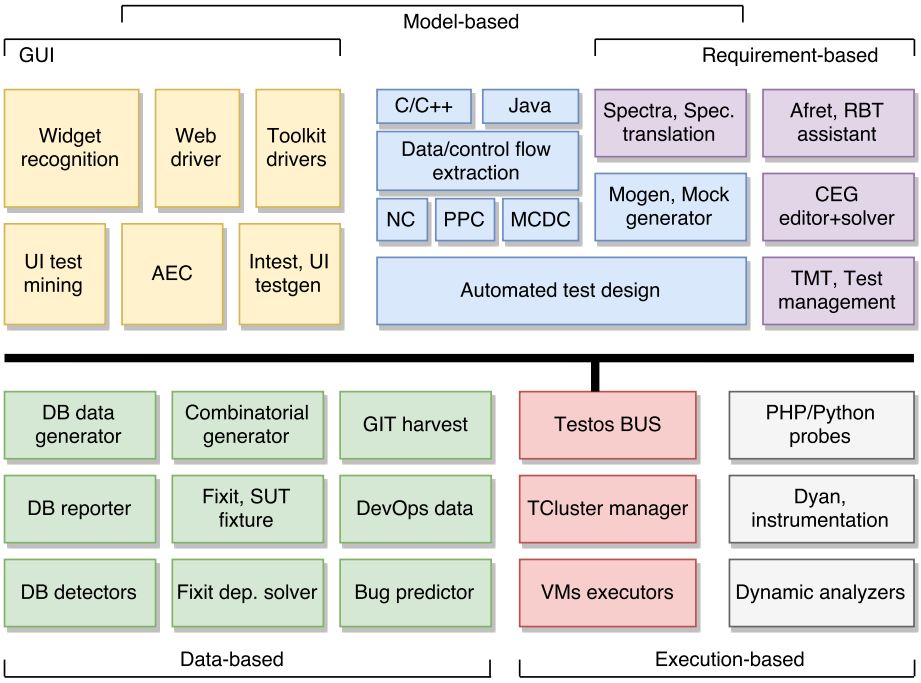
Výsledný program by teda mal byť ľahkou obálkou nad platformou Docker abstrahujúcou užívateľa od používania samotného Dockeru cez jeho vlastné užívateľské prostredie a namiesto toho poskytovať rozhranie, ktoré môže tester využívať bez nutnej znalosti Dockeru. Tým pádom môže spoľahnúť, že pre neho bude automaticky vytvorené testovacie prostredie vo forme kontajneru a že sa môže sústrediť iba na nasadzovanie vlastných testov.

Hlavný prínosom práce by malo byť zjednodušenie automatického testovania grafických používateľských rozhraní aplikácií. Zjednodušený postup pri vytváraní kontajnerov v kombinácii s podporou grafického rozhrania môže urýchliť nasadzovanie kontajnerov, pomocou ktorých sa vykonáva automatizované testovanie paralelne. To znamená, že testovanie jedinej grafickej aplikácie môže byť vykonávané na veľkom množstve kontajnerov, ktoré vyžadujú minimum systémových prostriedkov.

Práca v prvom rade čitateľovi poskytuje náhľad do postupov testovania softvéru, na ktoré sa má výsledný program aplikovať. Ďalšia kapitola opisuje technológie, ktoré boli pri riešení využité na splnenie požiadaviek (napr. Docker) na výsledný program. v nasledujúcej kapitole sa čitateľ oboznámi s návrhom riešenia programovej časti z pohľadu ako už jeho štruktúry tak aj prepojenia jednotlivých častí. v ďalšej kapitole je implementácia programu popísaná z pohľadu programátora. Posledná kapitola čitateľa oboznámi s postupom testovania výsledného programu.

1.1 Platforma Testos

Testos (Test Tool Set) [11] je projekt, ktorého hlavným cieľom je vytvorenie sady nástrojov podporujúcich automatizované testovanie softvéru. Nástroje v platforme Testos (viz. obrázok 1.1) kombinujú rôzne úrovne testovania a je možné ich radiť do niekoľkých kategórií: testovanie založené na modeloch (Model-based), testovanie založené na požiadavkách (Requirement-based), testovanie grafického užívateľského rozhrania (GUI), testovanie založené na dátach (Data-based) a dynamická analýza (Execution-based).



Obr. 1.1: Diagram schémy platformy Testos

V aktuálnom vývoji nástrojov pre dynamickú analýzu programov sú nástroje pre analýzu použitia knižničných volaní v binárnych programoch [4], nástroje pre monitoring paralélnych kontraktov v jazyku Java [3] a nástroj pre tvorbu kontajnerov pre stálosť testov, ktorý je zároveň touto prácou, vyvíjaný pod menom Fixit.

Kapitola 2

Použité postupy a technológie

Táto kapitola popisuje základy testovania softvéru. Zároveň sú niektoré postupy ako napríklad ustálené prostredia pre testovanie popísané bližšie, aby čitateľ pochopil ich koncept a ich aplikáciu v tejto práci. Ďalej čitateľa oboznamuje s technológiami, ktoré boli použité pri implementácii tejto práce. Zaoberá sa primárne virtualizáciou pomocou Linuxových kontajnerov a ďalej technológiami VNC a SSH.

Hlavnou súčasťou implementácie, už zo zadania, je projekt Docker, ktorý bol využitý na správu kontajnerov.

Aj keď si to často vývojári softvérových aplikácií nepripustia, človek robí chyby a faktom je, že softvér je chybný tiež. Primárnou úlohou testovania sotvéru je detekovať chyby v jeho implementácii, aby bolo možné odhaliť defekty a opraviť ich a týmto spôsobom zvýšiť jeho kvalitu. Testovanie však nezaručuje, že testovaný program bude bežať správne za akýchkoľvek podmienok - dokazuje iba to, že je chynbý za určitých konkrétnych podmienok.

V roku 2007 bola vydaná kniha [5] zaoberajúca sa automatizovanými testovacími postupmi pomocou svetovo populárneho frameworku[1] slúžiaceho na unit testing[2]. Meszaros vo svojej knihe postup automatizovaného testovania rozdeľuje na 4 fázy:

- **Set up** - nastavenie stáleho prostredia[3] do konfigurácie, ktorá je potrebná, aby SUT[4] vykazoval očakávané správanie a zároveň aj nastavenie všetkých závislostí potrebných na to, aby bol výsledok testov pozorovateľný,
- **Excercise** - spustenie SUT,
- **Verify** - kroky potrebné na vyhodnotenie, či bol očakávaný výsledok dosiahnutý,
- **Teardown** - vrátenie prostredia do pôvodného stavu, teda stavu pred fázou Set Up. Hlavným zámerom je odstrániť všetky zmeny, ktoré mohli vzniknúť ako vedľajšie účinky testovania.

[1] Framework je štruktúra poskytujúca základnú funkcionalitu pre používateľa, ktorý ju môže upraviť na tvar špecifický jeho programu

[2] Unit testing je postup testovania, ktorého úlohou je testovať fragmenty kódu, ktoré popisujú chovanie programu (funkcia, trieda, atď.) na odhalenie chýb na najnižšej úrovni a zníženie rozsahu testovania na vyšších úrovniach

[3] Stále prostredie (z angl. Test fixture) sú všetky veci potrebné na spustnie testu s očakávaným výsledkom

[4] SUT - System-Under-Test, alebo Software-Under-Test je systém (napríklad nejaký program), ktorý je testovaný

2.1 Verifikácia softvéru

Verifikácia softvéru má za cieľ dosiahnuť, že softvér, ktorý je vyvíjaný spĺňa všetky požiadavky, ktoré sú naň kladené a plní účel, pre ktorý bol vyvinutý. Existuje viacero prístupov ku verifikácii softvéru a to:

- Statická analýza - skúmanie vlastností softvéru bez jeho skutočného vykonania a to inšpekciou jeho zdrojového kódu,
- Dynamická analýza - skúmanie vlastností softvéru jeho vykonávaním, pričom sa skúma správanie programu za jeho behu. využitie stálych prostredí pre testovanie zapadá práve do tejto kategórie, pretože vytvárajú obal, v ktorom sa SUT spustí.

2.2 Stále prostredie pre testovanie

Vývoj softvéru v dnešnej dobe zahŕňa testovanie pomocou rozsiahlych automatizovaných testovacích sád (z angl. test suite, teda množina testovacích prípadov). Ak by však mali testeri pri každom behu testovacích sád nastavovať všetky náležité systémové závislosti, efektivita ich práce by sa drasticky znížila. Úlohou stálych prostredí v testovaní je zefektívnenie nasadení testovacích sád tým, že zavádzajú všetky tieto závislosti podľa pripravených postupov.

Za stále prostredie môžme považovať napríklad:

- nahranie predom známych dát do databázy pred tým, než ju začneme testovať,
- preinštalovanie operačného systému na počítači a práca s úplne čistým systémom,
- použitie známych súborov ako vstupné dáta do programu.

Na organizáciu testovacích prostredí existuje viacero stratégií [5], ktoré sú využiteľné v rôznych situáciach:

- Minimálne prostredie (angl. Minimal fixture) - pre každý test je vytvorené najmenšie možný prostredie. Test, ktorý využíva toto prostredie je vďaka minimalizácii informácií jednoduchší na pochopenie, avšak snaha potrebná na jeho vytvorenie je väčšia ako pri napríklad štandardnom prostredí.
- Štandardné prostredie (angl. Standard fixture) - tento prístup je vhodný v prípade, že prostredie je nadizajnované a vytvorené ešte pred vznikom samotných testov, na ktoré bude slúžiť.
- Čerstvé prostredie (angl. Fresh Fixture) - dizajn a skladba takéhoto prostredia sú koncipované tak, že je prostredie vo výsledku aplikované iba na jeden beh jediného testu. Prostredie sa použije pri spustení jedného konkrétneho testu a po jeho skončení je zrušené. Tento prístup je vhodný, ak chceme testovať v čistom stave. Čerstvé prostredia môžme ďalej rozdeliť na:
  - Prechodné (Transient) - prostredie vzniká s testom a po jeho skončení zaniká,
  - Perzistentné (Persistent) - prostredie existuje pred, aj po uskutočnení testu.
- Zdieľané prostredie (Shared Fixture) - životnosť takéhoto prostredia presahuje beh jedného testu. Je využívané viacerými testami bez toho, aby medzi nimi zaniklo a bolo nasadené odznova.

2.3 Linuxové kontajnery

Linuxový kontajner (LXC) [13] [10] je metóda virtualizácie na úrovni operačného systému používaná na spustenie instancie linuxového systému, ktorý je izolovaný od svojho hostiteľského systému. Zjednodušene by sa dalo povedať, že kontajner je Linuxový proces, alebo množina procesov, ktoré majú špecifické vlastnosti a sú spustené v izolovanom prostredí nakonfigurovanom hostiteľským systémom. Narozdiel od známejších virtuálnych strojov (VM) kontajnery nereplikujú celý operačný systém, ale iba jednotlivé komponenty, ktoré potrebujú, ako napríklad konkrétne knižnice, alebo binárne súbory. Služby jadra sú teda čerpané z hostiteľského systému, no pri používaní sa kontajnery javia ako samostatné, sebestačné systémy – simulujú vlastné sieťové rozhranie, vlastný strom procesov, nové používateľské účty, atď.

Najväčším prínosom kontajnerizácie aplikácií je, že sa dajú do kontajneru zabaliť so všetkými pre nich potrebnými závislosťami. Takto zabalená aplikácia sa teda dá spustiť vždy rovnakým spôsobom, nezávisle od prostredia, v ktorom je daný kontajner spustený. Vyplýva z toho veľa výhod pri vývoji softvéru v zmysle, že zabalenie programu zbavuje vývojárov problémov pri prenose medzi rôznymi systémmi. Uľahčuje tiež nasadenie nového systému elimináciou nutnosti opakovaných inštalácií konfigurácie.

Ďalšou výhodou kontajnerov je možnosť spustenia množstva kontajnerov v rámci jediného hostiteľského systému vzhľadom na ich nízke hardvérové nároky a fakt, že so systémovými prostriedkami nakladajú dynamicky.

2.3.1 Riadiace skupiny

Kontajnerizácia linuxových systémov má počiatky v koncepte zvanom riadiace skupiny (angl. control groups, alebo skrátene "cgroups") [7]. Riadiace skupiny dávajú k dizpozícii mechanizmus pre agregáciu, alebo partíciu skupiny úloh a ich budúcich potomkov do hierarchicky zoradených skupín so špecifikovaným chovaním. Podobne ako u procesov, potomkovské control groups dedia niektoré z atribútov ich predkov. Narozdiel od procesov však hierarchia riadiacich skupín netvorí jediný strom (v Linuxe sú všetky procesy potomkovia procesu *init*, ktorý je spúšťaný jadrom systému v rámci zavádzacej sekvencie systému a ktorý spúšťa ďalšie procesy), ale môže tvoriť mnoho stromových hierarchií v jedinom systéme, ktoré sú od seba nezávislé. Dôvodom takejto skladby je pridelenie každej takejto hierarchie jednému, alebo viacerým *subsystémom*, ktoré reprezentujú jeden zo systémových prostriedkov.

Riadiace skupiny dávajú používateľovi možnosť alokovať prostriedky (napríklad procesorový čas, systémová pamäť, šírku pásma pre sieť, atď.), alebo ich kombináciu medzi ním definované skupiny úloh (procesov), ktoré su v systéme spustené. Používateľ ďalej môže tieto skupiny monitorovať, zakázať skupinám prístup k istým prostriedkom a dokonca aj dynamicky meniť konfiguráciu cgroups v už bežiacom systéme, čo celkovo zvyšuje efektivitu vykonávania združených úloh. Agregáciou procesov sa znížia nároky na systémové prostriedky (procesor, operačná pamäť, sieť, atď.), čo zvýši výkonnosť a rýchlosť ich vykonávania.

2.3.2 Alternatívy ku Linuxovým kontajnerom

Táto sekcia stručne popisuje vybrané technológie podobné Linuxovým kontajnerom. Alternatív ku kontajnerom existuje viac a toto nie je ich úplný výčet.

Solaris Zones

Oracle Solaris Zones [6], skôr nazývané Solaris Containers sú súčasťou operačného systému Oracle Solaris 10. Zones izolujú softvérové aplikácie a služby pomocou voľne definovateľných hraníc. Zone je samostatné prostredie vytvorené vnútri jedinej isntancie operačného systému Oracle Solaris a zabezpečuje spustenie programu vnútri, v nezávislosti od vonkajšieho systému, pričom jednotlivé zóny pôsobia ako keby na nich bežal ich vlastný systém.

BSD Jails

BSD Jails [12], dostupné v Unixovom systéme FreeBSD stavajú na koncepte **chroot**, ktorý mení koreňový adresár nejakého procesu a tým ho efektívne izoluje od zvyšku systému tak, že nemá prístup mimo jemu prideleného koreňového adresára. Jails narozdiel od chroot prostredia virtualizujú prístup do súborového systému, používateľom a sieťovému podsystému.

Virtualizácia pomocou hypervízie

Poslednou zmienenou alternatívou je virtualizácia s hypervisingom. Hypervisor je funkcia, ktorá abstrahuje virtuálny operačný systém a jeho aplikácie od jeho hostiteľského operačného systému a teda vytvára samostatnú instanciu jadra virtualizovaného operačého systému. Týmto spôsobom je hostiteľ schopný umožniť viacerým virtuálnym strojom zdieľať jeho hardwarové prostriedky. Výhodou virtuálnych strojov s hypervisorom je najmä nezávislosť ich operačného systému od jadra hostiteľského. Narozdiel od kontajnerov sú schopné spustiť akýkoľvek operačný systém, nedbajúc pritom na jadro hostiteľského systému. Takto teda môžme spustiť napríklad systém Windows ako virtuálny stroj pod Linuxovým systémom.

Asi najznámejšími platformami implementújúcimi virtualizáciu s hypervíziou sú Oracle VM VirtualBox, alebo Microsoft Hyper-V.

2.4 Využite linuxových kontajnerov pre účely testovania softvéru

Ako už bolo spomenuté, kontajnery majú veľmi veľkú výhodu v tom, že sú malé, jednoducho prenositeľné obrazy operačného systému. Pre testera to tvorí prínos v zmysle, že si vnútri kontajneru môže vytvoriť stabilné testovacie prostredie bez nutnosti ovplyvňovania vlastného systému. Kontajner tak môže opakovane použiť a nemusí riešiť problémy, ako zmeny knižníc, systémových závislostí, alebo balíčkov, ktoré ovplyvňujú SUT, na ktorom tester v tej chvíli pracuje.

Ak pri práci v kontajneri nájde tester v softvéri chybu a pozná aj spôsob, akým ju reprodukovať, oznámi vývojárom, že nastala chyba pri spustení programu a opíše postup, ktorým k nej došiel. Pri takomto jednaní však môže nastať problém, keď sa vývojárovi nepodarí na jeho systéme popísanú vadu nájsť. Môže to byť spôsobené rozdielmi v systémových závislostiach na počítači testera a na stroji vývojára.

Kontrolovať rozdiely medzi danými systémami by bolo vo väčšine prípadov časovo náročné a volať vývojára ku počítaču testera pri výskyte každej takejto chyby zas nepraktické.

Využitie virtualizácie je v takýchto prípadoch najvýhodnejšou cestou. Virtualizácia s hypervíziou je však problematická z dôvodu veľkosti jednotlivých obrazov systému a využitie Jails, alebo Solaris Zones je nepraktické vzhľadom na ich exkluzivitu.

V tomto momente do situácie vstupuje možnosť využitia linuxových kontajnerov.

Za použitia kontajneru je tester schopný SUT efektívne zabaliť. Pokiaľ sa testerovi podarí zreprodukovať chybu vnútri kontajneru, môže ho celý zabaliť so všetkými závislosťami, ktoré vo vzniku danej chyby hrali rolu. Namiesto zdĺhavého opisu postupu, ako sa dostať k ním nájdeným výsledkom môže kontajner odovzdať vývojárovi, ktorý si ho na svojom počítači spustí a uvidí zreprodukovanú chybu na vlastné oči. Môže teda zistiť, čo je zdrojom chyby a čo ju spôsobilo priamo v systéme, kde bola nájdená.

V rámci stratégií pre koncepciu ustálených testovacích prostredí zmienenú v sekcii 2.2 sa dá kontajnerizácia aplikovať na ktorúkoľvek:

- Štandardné prostredie - vytvorí sa kontajner obsahujúci všetky potrebné závislosti a využije sa na priebežné testovanie.
- Minimálne prostredie - toto prostredie by sa dalo realizovať napríklad kontajner so základným obrazom Alpine Linux. Takýto kontajner zaberá na disku okolo 8 MB a obsahuje iba najnutnejšie závislosti potrebné na funkčnosť systému.
- Čerstvé prostredie - v tomto prípade by bolo možné vytvárať automatizovaný test, ktorého súčasťou by bolo postavenie kontajneru, otestovanie SUT a následné vymazanie kontajneru.
- Zdieľané prostredie - tu by prostredie mohol predstavovať konštantne spustený kontajner povedzme bežiaci na serveri, na ktorom by boli opakovane spúšťané testy.

2.5 Projekt Docker

Docker [2] je open-source platforma slúžiaca na nasadzovanie programov vnútri softvérových kontajnerov vyvíjaná spoločnosťou Docker Incorporated. V súčasnosti Docker spolupracuje so spoločnosťami, ak napríklad Red Hat, alebo Microsoft a podporuje operačné systémy Linux a Windows so 64 bitovou architektúrou.

Ako väčšina open-source projektov v sebe spája už existujúce zaužívané Linuxové technológie, ako napríklad *copy-on-write* (zdieľanie systémových prostriedkov, kedy sa duplikovaný prostriedok zdieľa medzi kópiou a originálom, pokiaľ sa nezmenil) súborové systémy a Linuxové kontajnery a obohacuje ich súčasťami, ktoré sú zamerané pre softvérových vývojárov ako napríklad:

- Verzovací systém - používateľ je schopný uložiť si kontajner s aktualizovaným stavom bez nutnosti ovplyvnenia počiatočného stavu,
- Prenositeľnosť nasadenia - kontajner je možné jednoducho zabaliť a preniesť do iného systému, ktorý podporuje Docker,
- Jednoduché nasadenie kontajneru pomocou *Dockerfile*,
- Jednoduché nasadenie multikontajnerovej aplikácie pomocou nástroja **Docker Compose**,
- Znovupoužiteľnosť komponentov - možnosť vytvorenia si základného obrazu, tvoriaceho dokonalé prostredie a následné nasadenie rôznych aplikácií do tohto prostredia.

Docker oplýva bohatou dokumentáciou a tak pri implementácii práce nebol problém dohľadať akékoľvek informácie, týkajúce sa či už funkcionality kontajnerov tak aj chybách, ktoré sa v platforme prejavili. Na internete sa tiež nachádza množstvo príspevkov napísaných úžívateľmi Dockeru, ktoré boli pri porozumení niektorých funkcií, alebo implementačných postupov veľmi nápomocné.

2.5.1 Dockerfile

Docker je schopný vytvoriť obraz systému automatizovane, za použitia tzv. **Dockerfile** [1]. Dockerfile je súbor, ktorý obsahuje predpis v podobe príkazov, ktoré by inak používateľ zadával do príkazového riadku.

Formát jednej inštrukcie je:

```
INSTRUCTION arguments
```

kde `INSTRUCTION` je inštruckia špecifická pre Dockerfile a `arguments` sú argumenty danej inštrukcie. Tieto argumenty môžu byť viacriadkové, oddelením pomocou znaku spätného lomítka. Každý Dockerfile musí začínať inštrukciou `FROM`, ktorá špecifikuje *základný obraz*, čo je obraz systém, na ktorý sa ďalej nabaľuje konfigurácia špecifikovaná v Dockerfile. Ako základný obraz môže slúžiť základná inštalácia systému, ale aj užívateľom predpripravená varianta určená pre jeho potreby.

Pomocou príkazu `docker build` užívateľ spustí stavbu obrazu z Dockerfile a kontextu. Kontext sú v podstate všetky súbory nachádzajúce sa na ceste danej ako:

- cesta v súborovom systéme hostiteľa
- url adresa Git repozitára na internete

Kontext je spracovávaný rekurzívne a tak sa neodporúča podsúvať ako cestu objemné adresáre, ako napríklad `/`, teda koreňový adresár v Linuxových systémoch. Vyplýva to z toho, že proces Docker build proces nebeží v príkazovom riadku, ale v *Docker démone* [5]. Nastavenie kontextu na koreňový adresár by tým pádom znamenalo, že by sa celý obsah pevného disku presunul do pamäti Docker démona.

Aj napriek tomu, že sa príkazy v Dockerfile vykonávajú jeden za druhým, vo výsledku sa medzi sebou svojimi dočasnými nastaveniami neovplyvňujú. Pre príklad uvažujme tento krátky príkaz v Dockerfile:

```
RUN cd /tmp
```

Tento príkaz hovorí, aby sa kontext príkazového riadku zmenil na priečinok **/tmp**. Táto inštrukcia však na nasledujúce inštrukcie nebude mať žiaden vplyv. Tento fakt vyplýva zo spôsobu, akým Docker build pracuje pri zostavovaní výsledného obrazu. Po vykonaní každej inštrukcie v Dockerfile si Docker démon totiž ukladá zmeny do nového obrazu, ak je to nutné a až potom užívateľovi vráti ID výsledného obrazu. Tento mechanizmus sa využíva pri opätovnom spustení Docker build, kedy sa takéto prechodné image nachádzajúce sa v cache Dockeru používajú na urýchlenie procesu zostavovania image. v jednoduchosti, ak používateľ upraví Dockerfile, prechodné obrazy, na ktoré boli aplikované príkazy nachádzajúce sa v Dockerfile pred upraveným príkazom, sú načítané z cache a pre inštrukcie za úpravou sa vytvoria nové prechodné obrazy.

[5] dockerd, alebo Docker démon je perzistentný proces, ktorý je spustený v pozadí systému a stará sa o správu kontajnerov

Pri realizácii programovej časti tento mechanizmus niekedy spôsoboval problémy pri inštalácii balíkov z oficiálnych repozitárov pomocou správcov balíkov. Vzhľadom na to, že inštrukcia slúžiaca na obnovovanie zoznamov balíkov bola na začiatku Dockerfile, Docker build ju nerealizoval a balíkový správca kontajnerizovaného systému nedokázal požadované zdroje nainštalovať. Pri vývoji bol teda používaný prepínač `--no-cache`, ktorý zakázal využitie prechodných obrazov, čím bolo docielené, že obraz sa zostavoval od začiatku.

2.6 Vzdialená plocha VNC

Vzdialená plocha VNC (Virtual Network Computing) [9], je systém umožňujúci zdieľanie pracovnej plochy počítača v grafickom režime za účelom jeho ovládania. VNC na toto ovládanie využíva protokol *RFB (Remote FrameBuffer)* [8], ktorý slúži na vzdialený prístup ku grafickým užívateľským rozhraniam.

Pomocou siete prenáša udalosti myši a klávesnice na ovládaný počítač a spätne zas zmeny grafickej obrazovky ovládaného počítača. Vďaka jednoduchému princípu RFB protokolu VNC nie je závislé na akomkoľvek systéme, čo znamená, že klient, ktorý beží na systéme Linux sa pomocou VNC dokáže pripojiť na systém Windows bez akejkoľvek dodatočnej konfigurácie alebo adaptéra.

Architektúra VNC sa skladá z troch častí:

- **Protokol VNC** (teda RFB).
- **Server VNC** je program spustený na počítači, ktorý je ovládaný. Server teda pasívne povoľuje klientovi pripojiť sa na neho. Hodný zmienky je aj fakt, že VNC server nevyžaduje fyzické zobrazovacie zariadenie, čo tvorí výhodu v kombinácii s kontajnermi, ktoré natívne grafické užívateľské rozhranie nepodporujú. Úlohou serveru je klientovi posielať výrezy jeho rámovej vyrovnávacej pamäte[6]. Server teda pasívne povoľuje klientovi pripojiť sa na neho.
- **Klient VNC** je program, ktorý má na starosti všetku interakciu so serverom, čo zahrňuje jeho sledovanie a ovládanie.

Hodný zmienky je aj fakt, že server VNC nevyžaduje fyzické zobrazovacie zariadenie, čo tvorí výhodu v kombinácii s kontajnermi, ktoré natívne grafické užívateľské rozhranie nepodporujú.

2.7 Secure Shell (SSH)

Secure Shell [14] je kryptovaný sieťový protokol slúžiaci na vzdialené prihlasovanie a riadenie sieťových služieb zabezpečene na nezabezpečenej sieti.

SSH sa skladá z:

- protokol SSH,
- server SSH - program spustený ako démon na pozadí systému, ktorý prijíma vzdialené pripojenia,

[6] Rámová vyrovnávacia pamäť (angl. frame buffer) - fragment pamäte, ktorý obsahuje obraz, ktorý sa má zobraziť na obrazovke

- klient SSH - program, ktorý využíva protokol SSH na pripojenie ku vzdialenému počítaču,

SSH sa zvyčajne používa na prihlásenie do vzdialeného systému a spúšťanie rôznych príkazov, no podporuje napríklad aj:

- Tunneling - Dáva používateľovi možnosť pristupovať ku sieťovej službe, ktorú nižšie položená vrstva siete nepodporuje.
- Pripojenie s podporou X11 - Schopnosť spustiť aplikáciu s grafickým používateľským rozhraním priamo na serveri.
- Prenos súborov pomocou *SSH file transfer (SFTP)* a *Secure Copy (SCP)*.

Kapitola 3

Návrh riešenia

Táto kapitola čitateľovi popisuje návrh riešenia práce z architekturálneho a behaviorálneho hľadiska. Poskytuje náhľad do spôsobu, akým jednotlivé časti riešia ich problematiku a taktiež ako tieto časti spolupracujú.

Programovú časť zastupovali skripty napísané pre Python 3.6 a Unix Shell. Pre súbory obsahujúce konfiguráciu v textovej podobe bol zvolený formát YAML.

3.1 Špecifikácia požiadaviek na výsledný program

Systém umožňuje vytvoriť kontajner, ktorý obsauje:

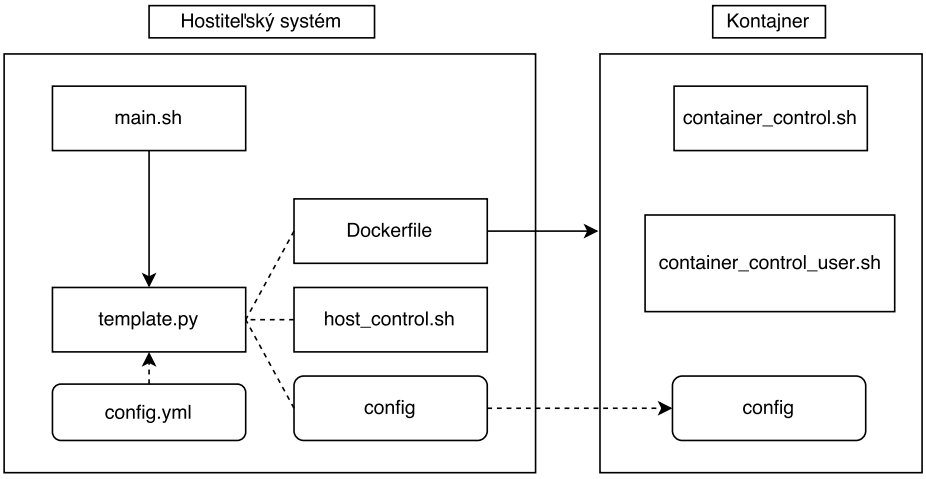
1. balíky zadané užívateľom, pokiaľ sú obsiahnuté v repozitároch danej distribúcie,
2. spustený server SSH,
3. spustený server VNC a ďalej:
   (a) rozlíšenie serveru VNC je dané používateľom,
   (b) obsahuje správcu okien,
   (c) je schopný vytvoriť snímok obrazovky obrazovky serveru VNC a to voliteľne aj s oneskorením zadaným užívateľom,
4. užívateľa bez supervízorských práv špecifikovaného používateľom,
5. súbory špecifikované užívateľom nakopírované z hostiteľského systému,
6. zdieľané médium (adresár) s hostiteľským systémom.

Systém tiež musí dokázať spustiť kontajner z vytvoreného obrazu spolu s užívateľom dodanými testami.

3.2 Architektúra a správanie

Vzhľadom na využite kombinácie viacerých programovacích jazykov a platformy Docker sa musel program rozdeliť na viacero spolupracujúcich častí.

Obrázok 3.1 zjednodušene ukazuje zloženie programu. Na ľavej strane diagramu je zobrazený hostiteľský systém. Počiatočnými súbormi programu sú:



Obr. 3.1: Zjednodušený diagram architektúry programu ukazujúci postup tvorby Docker obrazu.

- **main.sh** - hlavný riadiaci Shell skript, ktorý spúšťa ďalšie skripty,
- **template.py** - skript napísaný v jazyku Python, slúžiaci na vygenerovanie ďalších skriptov a súborov (viac v sekcii 3.4),
- **config.yml** - konfiguračný súbor vo formáte YAML využívaný skriptom template.py (viac v sekcii 3.3).

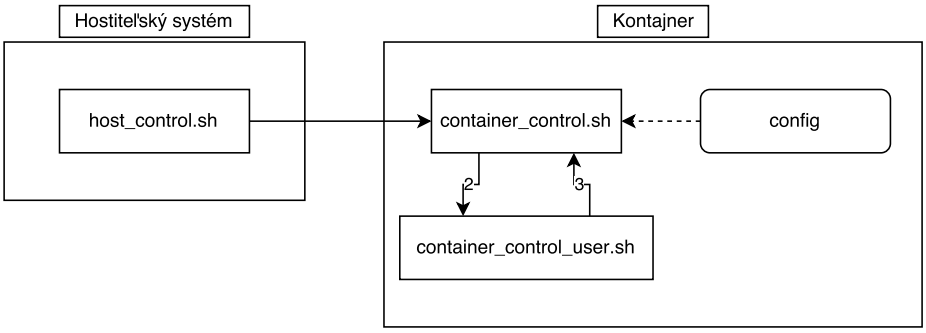
Tento diagram ukazuje jednotlivé kroky správania programu a prepojenie súborov:

1. Najvyšší riadiaci skript **main.sh** spustí Python skript **template.py**.
2. Skript **template.py** načíta konfiguráciu z YAML súboru (znázornené prerušovanou hranou). **config.yml** a vygeneruje súbory **Dockerfile**, **host_control.sh** a **config**
3. Po skončení skriptu **template.py** sa spustí skript **host_control.sh**, ktorý spustí príkaz `docker build`, ktorý postaví Docker obraz podľa predpisu v Dockerfile (znázorňuje to hrana vedúca od Dockerfile ku kontajneru). v priebehu zostavovania sa do kontajneru nakopírujú riadiace skripty a aj súbor **config**, ktorý zároveň slúži pre skripty vnútri kontajneru.

Rozsiahlejší opis tohto postupu sa nachádza v sekciách 3.3, 3.4 a 3.5 tejto kapitoly. Tento diagram tiež zobrazuje len súbory nutné na dosiahnutie základnej funkcionality programu.

Program ďalej pokračuje spustením kontajneru vytvoreného v predchádzajúcich krokoch. Na diagrame 3.2 sú hrany označené číslami, ktoré znázorňujú tieto kroky:

1. Riadenie programu sa presunie zo strany hostiteľského systému na stranu kontajneru. Tu má za úlohu skript **container_control.sh** inicializovať všetky potrebné nastavenia, pričom si vlastnú konfiguráciu čerpá zo súboru **config**.
2. Skript **container_control.sh** po vykonaní svojich úkonov zavolá skript **container_control_user.sh**, ktorý obsahuje inštrukcie zadané testerom.
3. Po skončení vykonávania skriptu **container_control_user.sh** sa riadenie vráti pod réžiu **container_control.sh**, ktorý vykoná finálne inštrukcie a skončí.



Obr. 3.2: Zjednodušený diagram popisujúci spustenie kontajneru.

Detailnejšie túto časť postupu programu popisujú sekcie 3.5.3 a 3.5.3.

Po ukončení všetkých skriptov sa kontajner nevypína, ale zostáva spustený v pozadí, aby mal tester možnosť pripojiť sa na neho a pracovať s jeho systémom.

3.3 Konfigurácia skriptov

Najzákladnejšia užívateľsky programovateľná konfigurácia programu bola naimplementovaná pomocou súboru vo formáte YAML[1]. Užívateľ má možnosť nakonfigurovať položky ako napríklad:

- označenie základného Docker obrazu,
- meno spusteného kontajneru,
- špecifikácia súborov, ktoré majú byť nakopírované z hosťovského systému do kontajneru,
- či má obraz obsahovať server SSH, alebo server VNC,
- heslo pre používateľov a servery.

Tvoriaci Python skript template.py tento súbor prečíta za pomoci Python modulu yaml (pyyaml) a načítané dáta su uložené do pamäte. Tieto dáta sú následne využité pri tvorbe ďalších súborov.

3.3.1 Správcovia balíkov

keďže implementácia zahŕňala možnosť nastaviť distribúciu linuxového systému podľa voľby základného Docker obrazu, vyplývala nutnosť počítať s tým, že jednotlivé vetvy distribúcií využívajú rozdielne balíkové repozitáre a rôznych balíkových správcov. Program podporuje správcov balíkov:

- **apt** - Deriváty systému Debianu,
- **dnf** - Fedora,

[1]YAML (YAML Ain't a Markup Language) - ľudsky prečítateľný formát, ktorý slúži na serializáciu dát a najčastejšie sa využíva na konfigurácie programov

- **pacman** - Deriváty systému Arch Linuxu,
- **apk** - Alpine Linux.

Keďže zrejme neexistuje jednoduchý, priamočiary spôsob ako podľa mena Docker obrazu zistiť meno balíkového manažéra, povinnosťou používateľa je tiež pri konfigurácii vyplniť meno správcu, ktorým daná distribúcia disponuje. Program si podľa daného správcu zvolí korešpondujúce príkazy pre inštaláciu jednotlivých balíkov, ktoré sa inštalujú pri tvorbe obrazu z Dockerfile.

Ako už bolo spomenuté v časti popisujúcej Dockerfile (sekcia 2.5.1), pri testovaní programu bolo odhalené, že príkaz na osvieženie repozitárov nemal vždy požadovaný efekt. Ak sa v Dockerfile príkaz na osvieženie (napríklad apt update) a príkaz na inštaláciu balíku (apt install) vyskytovali v samostatných Dockerfile `RUN` inštrukciách, príkaz na osvieženie nemal efekt a tak nedokázal nájsť požadovaný balík na inštaláciu. s najväčšou pravdepodobnosťou je dôvodom tejto chyby spôsob, akým si Docker vytvára priebežné obrazy po každej inštrukcii v Dockerfile. Platforma Docker na tento problém neposkytuje priame riešenie a teda je pri tvorbe Dockerfile pred každý príkaz na inštaláciu vložený príkaz na osvieženie repozitárov. Týmto spôsobom sa nemôže stať, že sa bude balíkový správca pokúšať o stiahnutie pomocou zastaralých záznamov.

3.4 Tvorba Dockerfile, konfiguračných súborov a riadiacich skriptov

Vzhľadom na fakt, že mnoho premenných v programe záleží od nastavenia používateľa, niektoré súbory bolo nutné generovať podľa konfigurácie v priebehu programu. Tieto súbory sú:

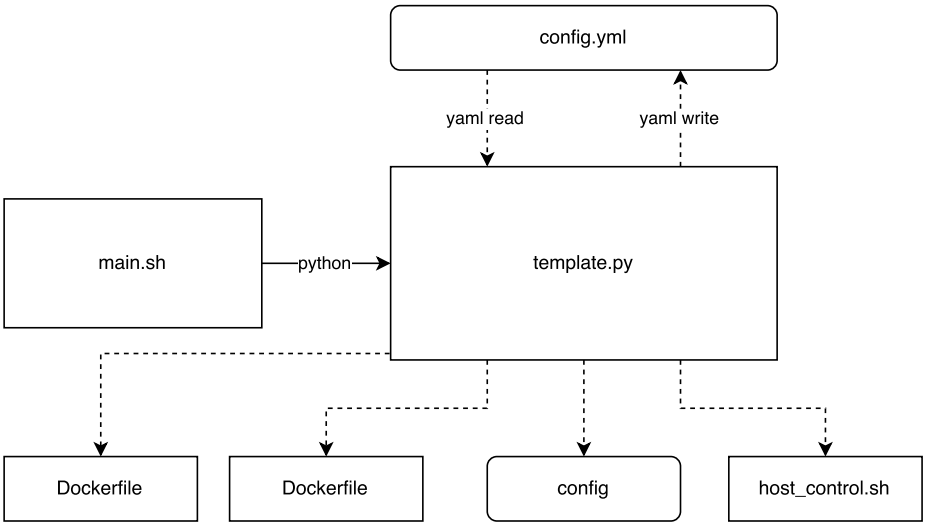
- `container_control.sh` - riadiaci Shell skript, ktorý slúži na ovládanie činnosti programov vnútri kontajneru,
- `Dockerfile` - Dockerfile, podľa ktorého sa postaví požadovaný obraz systému,
- `config` - konfiguračný súbor využívaný shell skriptami a
- `host_control.sh` - Shell skript, ktorý prevezme riadenie od skriptu main.sh.

Obrázok 3.3 názorne ukazuje postup vytvorenia týchto súborov.

Pri spustení skriptu main.sh ja najprv volaný interpret Python nad skriptom template.py. Implicitne sa pri spustení tvoriaceho Python skriptu spúšťa interaktívny dialóg, ktorý užívateľa prevedie krok za krokom jednotlivými položkami konfigurácie a vytvorí konfiguračný súbor, z ktorého potom tvoriaci skript načítava požadované nastavenie. Hrany v diagrame 3.3 označené ako **yaml read** a **yaml write** znázorňujú tento postup.

Užívateľ má tiež možnosť vynechať interaktívny dialóg. Pomocou argumentu príkazového riadku `--no-dialogue` predanému tvoriacemu Python skriptu sa interaktívny dialóg preskočí a prebehne načítanie priamo zo súboru config.yml. Tento postup je vhodný v prípade, že má tester tento konfiguračný súbor pripravený dopredu a obsahuje nastavenie, ktoré je vyskúšané.

Po spracovaní konfiguračných dát skript nakoniec vytvorí predom zmienené súbory, ukončí sa a riadenie preberá skript host_control.sh.



Obr. 3.3: Tvorba Dockerfile, konfiguračných súborov a riadiacich skriptov.

3.4.1 Všeobecné heslo

Položka v konfiguračnom súbore config.yml general-password uchováva reťazec, ktorý je následne využitý ako heslo pre server VNC, server SSH, superpoužívateľa a bažného používateľa.

3.5 Riadenie programu

Táto sekcia čitateľa podrobnejšie oboznámi ako program pracuje v jednotlivých krokoch a ako jeho súšasti komunikujú. Je v nej tiež popísané, akým spôsobom boli aplikované súčasti ako server SSH, alebo vzdialená plocha VNC.

3.5.1 Skript host_control.sh a stavba obrazu pomocou docker build

Po vytvorení všetkých potrebných súborov sa začne vykonávanie skriptu host_control.sh. Tento skript má v prvom rade za úlohu spustiť príkaz `docker build` nad súborom Dockerfile, ktorý bol vytvorený v predchádzajúcom kroku.

Docker build potom pri stavbe obrazu:

1. Nainštaluje požadované balíky pomocou správcu balíkov.
2. Nakopíruje konfiguračný súbor config, ktorý budú využívať Shell skripty.
3. Zmení heslo pre administrátorský účet na hodnotu zadanú používateľom (general-password).
4. Do súborového systému kontajneru nakopíruje súbory zadané používateľom.
5. v prípade, že sa má vytvoriť bežný, neadministrátorský účet, vykoná sa konfigurácia, bližšie popísaná v sekcii 3.5.1, v časti Vytvorenie používateľa.
6. v prípade, že si užívateľ v kontajneri želá mať server VNC, vykoná sa konfigurácia, bližšie popísaná v sekcii 3.5.1, v časti Aplikácia vzdialenej plochy VNC.

7. Ak chce mať užívateľ v kontajneri server SSH, vykoná sa konfigurácia popísaná v sekcii 3.5.1, v časti Aplikácia SSH.

8. Nakoniec sa nastaví spôsob, akým sa spúšťa kontajner, bližšie popísaný v sekcii 3.5.3.

Aplikácia vzdialenej plochy VNC

Ak si užívateľ vyžiada inštaláciu serveru vzdialenej plochy VNC v kontajneri, v prvom rade sa pomocou balíkového správcu do kontajneru nainštaluje príslušný balík. Ako bolo spomenuté v sekcii 3.3.1, rôzne Linuxové distribúcie využívajú rôzne balíkové manažéry a rôzne repozitáre. Tým pádom nie všetky repozitáre obsahovali totožný balík s implementáciou VNC serveru. Pre vyššie zmienené balíkové manažéry boli teda vybrané tieto balíky:

- apt - tightvncserver,
- dnf - tigervnc,
- pacman - tigervnc.

Repozitáre systému Alpine Linux balíkom obsahujúcim server VNC nedisponujú a tak táto práca neimplementovala možnosť spustenia serveru vzdialenej plochy v kontajneri, na ktorom beží Alpine. Tento fakt je uvážený aj pri konfigurácii programu pomocou interaktívneho dialógu (viz. sekcia 3.4) a ak je zvolený balíkový správca apk, možnosť nainštalovať VNC server je preskočená a hodnota v konfiguračnom súbore je nastavená na `false`.

Vďaka tomu, že program tigervnc je pôvodom vetva[2] programu tightvncserver, nemusel byť pri konfigurácii serveru uvážený alternatívny postup.

Prvým krokom inštalácie serveru VNC do kontajneru je nainštalovanie samotného balíku so serverom. Pre prípad, že by sa používateľ rozhodol vykonať snímok obrazovky VNC serveru sa taktiež inštaluje aj balík **scrot**, ktorý obsahuje jednoduchú utilitu na zachytávanie obrazovky (snímok obrazovky sa ukladá vo formáte PNG a meno výsledného súboru je *screenshot.png*).

Pri implementácii sa naskytla prekážka v konfigurácii serveru VNC, ktorý potreboval mať zvolené heslo. Ako finálny postup, aplikovaný vo výslednom programe bol zvolený skript, ktorý toto heslo automaticky nastaví. Utilita `vncpasswd` totiž disponuje interaktívnym dialógom, ktorý ma za úlohu nastaviť toto heslo, v prípade, že nenájde súbor ktorý ho obsahuje. Pri spustení kontajneru sa tiež spustí Shell skript **setup_vnc_pass.sh**. Tento skript zavolá `vncpasswd` a nastaví heslo pomocou programu `expect`, ktorý interaguje s dialógom a vypĺňa požadované hodnoty. z tohto dôvodu je pri inštalácii balíkov obsahujúcich VNC inštalovaný aj balík `expect` pre prípad, že by ho základný obraz neobsahoval.

Správca okien

Ak sa užívateľ rozhodne pre prítomnosť VNC serveru v kontajneri, je mu ponúknutá možnosť doinštalovať aj správcu okien. Hlavným dôvodom tejto súčasti je uľahčenie ovládania grafického používateľského rozhrania vnútri kontajneru. Ako balík obsahujúci správcu okien bol zvolený **fvwm**. Tento správca je v prvom rade jednoduchý a nevyžaduje množstvo systémových prostriedkov, čo je pre účely testovania optimálne. Zároveň podporuje súčasti ako napríklad podpora viacerých pracovných plôch, čo môže uľahčiť prácu s rozsiahlejšími testovacími sadami, alebo len kontajnerom samotným.

[2] Vetva (angl. branch) je v softvéri pojem, ktorý označuje projekt, ktorý bol v istom momente duplikovaný a od toho momentu vyvíjaný rozdielne, ako pôvodný

Aplikácia SSH

Ďalšou súčasťou programu je možnosť spustenia SSH serveru vnútri kontajneru. Táto možnosť je vhodná najmä v prípadoch, ak chce tester ovládať kontajnerizovaný systém bez využitia VNC, alebo príkazu `docker attach`[3].

Podobne ako server VNC, aj balík SSH serveru sa inštaluje v dobe zostavovania obrazu zo súboru Dockerfile. Kvôli rozmanitosti repozitárov bolo nutné podobne ako pri riešení serveru VNC rozlišovať balíky obsahujúce implementáciu SSH servera na základe správcu balíkov a to:

- apt - openssh-server,
- dnf - openssh-server,
- pacman - openssh,
- apk - openssh.

Vzhľadom na to, že všetky použité balíky obsahujú rovnakú implementáciu SSH serveru nebolo nutné pri spúšťaní a konfigurácii vytvárať špecifické postupy pre jednotlivé distribúcie.

V prípade, že užívateľ vyžaduje prítomnosť serveru SSH v kontajneri je v kroku stavby obrazu do súborového systému kontajneru nakopírovaný adresár obsahujúci SSH kľúče. Tento postup bol zvolený preto, že ak by sa pri každej konfigurácii kontajnerizovaného systému generovali nové kľúče nastal by problém pri pripájaní z hostiteľského systému, ktorý si pri prvom pripojení uložil kľúče a pri ďalšom spustení sa mu nezhodujú, keďže boli v kontajneri vygenerované nové.

Jediná prekážka, ktorú bolo nutné prekonať vznikla špecifickým nastavením balíka distribuovaného pomocou balíkového správcu apt. v jeho konfigurácii bola možnosť prihlásiť sa do systému pomocou SSH pod účtom superpoužívateľa implicitne zakázaná. Ako riešenie bol použitý editor **sed**[4], ktorý v konfiguračnom súbore vyhľadá položku nastavujúcu povolenie prihlásenia sa pod účtom superpoužívateľa a nastaví jej hodnotu na takú, ktorá tento úkon povoľuje. z tohto dôvodu sa pri inštalácii balíku SSH serveru inštaluje aj balík sed. v prípade, že by sa sed v systéme nenachádzal došlo by ku chybe pri zostavovaní obrazu a nebolo by možné spustiť požadovaný kontajner.

Vytvorenie používateľa

Pre testovanie bez administrátorských práv bola ako súčasť implementovaná možnosť vytvoriť bežného používateľa. Táto možnosť je nastaviteľná pri konfigurácii, kedy tester zadá meno požadovaného používateľa. Následne sa vytvorí jeho domovský priečinok a nakoniec samotný používateľský účet s heslom zadaným v premennej `general_password`. Pri jeho vytvorení je tento účet nastavený ako implicitný a kontajner sa ďalej spúšťa pod jeho prihlásením. Do tohoto účtu je tiež možné pripojiť sa pomocou SSH. Vytvorením bežného účtu pôvodný administrátorský účet nezaniká.

[3] Príkaz `docker attach` je súčasťou platformy Docker a slúži na pripojenie k terminálu v spustenom kontajneri

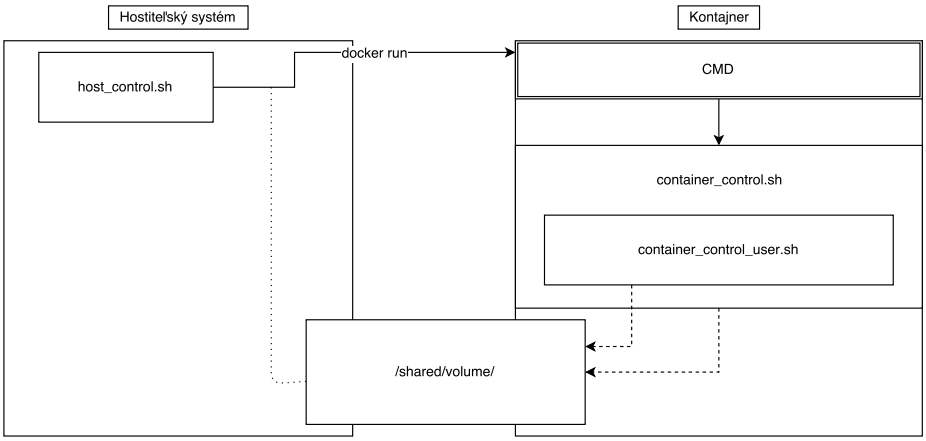
[4] **sed** (stream editor) je nástroj slúžiaci na úpravu vstupného toku dát pomocou základných textových transformácií na základe dodaného skriptu

3.5.2 Spúšťanie skriptov vnútri kontajnera

Pri prvotnej implementácii bolo nutnosťou nájsť spôsob, akým sa budú spúšťať Shell skripty, ktoré ovládajú činnosť v hostiteľskom systéme a následne vo vnútri kontajnerizovaného systému. Pri stavbe obrazu z Dockerfile sa do súborového systému kontajneru kopírujú riadiace skripty, ktoré vykonávajú testy na strane hosťovského systému. Otázkou ostávalo, ako vlastne spustiť tieto skripty vo vnútri kontajneru.

Prvá implementácia zahŕňala spustenie vykonávania skriptov pomocou príkazu `docker exec`, ktorý slúži na spustenie príkazu vo vnútri spusteného kontajneru (funguje podobne ako SSH). Testovacie skripty sa najprv nakopírovali do kontajneru pomocou príkazu `docker cp` a následne boli spustené prikazom `docker exec`. Pri dlhšom skúmaní však bolo zistené, že skripty spustené pomocou `docker exec` vyžadujú terminál a tak sa stávalo, že sa ich vykonávanie správne nedokončilo.

3.5.3 Skript container_control.sh a riadenie činnosti vo vnútri kontajneru



Obr. 3.4: Diagram znázorňujúci činnosť programu v kontajneri. Hrana označená ako "docker run"znázorňuje spustenie kontajneru a následné prebranie riadenia v kontajneri príkazom CMD a taktiež vytvorenie zdieľaného média medzi hostiteľom a kontajnerom. Prerušované orientované hrany znázorňujú výstup jednotlivých skriptov do tohoto zdieľaného média.

Ako bolo spomenuté v sekcii 3.5.1, pri skladaní Docker obrazu sa v poslednom kroku stavby nastaví spôsob, akým sa kontajner spúšťa. z dôvodu problému zmieneného v prechádzajúcej sekcii 3.5.2 musela byť implementácia zmenená na toto riešenie.

Vo finálnej verzii sa riadiace skripty pre kontajner kopírujú počas stavby obrazu z Dockerfile za využitia inštrukcie `COPY`. Samotné spustenie skriptov je zabezpečené pomocou Dockerfile inštrukcie `CMD`. Príkaz `CMD` nastavuje implicitné spustenie kontajneru. Riadiaci skript sa teda automaticky spustí pri každom spustení kontajneru bez nutnosti zadávania ďalších príkazov. Pri takomto spúšťaní sa chyba podobná chybe príkazu `docker exec` nevyskytla a zatiaľ je to najlepšie riešenie tejto problematiky.

Znamená to, že sa pri spustení kontajneru spustí skript **container_control.sh**. Tento skript má za úlohu riadiť všetky činnosti, ktoré sú vyžadované predchádzajúcou konfiguráciou.

Predpokladajme predpis kontajneru, ktorý má obsahovať všetky doteraz zmienené súčasti (VNC, SSH, ...). v takom prípade skript **container_control.sh**:

1. Načíta premenné zo súboru **config**.
2. Spustí démona serveru SSH.
3. Nastaví heslo serveru VNC
4. Zapne VNC server s rozlíšením, ktoré načítal zo súboru config.
5. Spustí užívateľský skript **container_control_user.sh** a vyčká na jeho dokončenie.
6. Počká požadovanú dobu a vytvorí snímok obrazovky, ktorý sa uloží do zdieľaného priečinku.

Skript container_control_user.sh

Primárny zámer tohto skriptu je spustenie samotných testov. Ako jediný súbor, ktorý nie je vygenerovaný za behu programu (neberúc v úvahu SSH kľúče a skript nastavujúci server VNC) je určený priamo testerovi, ktorého úlohou je vložiť do neho všetky úkony potrebné na to, aby sa požadované testy vykonali. z tohto dôvodu je od užívateľa vyžadovaná znalosť tvorby Shell skriptov.

Zdieľaný priečinok

Pre zjednodušenie prenosu súborov medzi hostiteľským systémom a kontajnerom bol zavedený postup, kde sa priečinok nachádzajúci v hostiteľskom systéme nazdieľa do súborového systému kontajneru. Tento postup tvorí výhodu v tom, že užívateľ nemusí kopírovať všetky súbory pri stavbe obrazu, alebo príkazom `docker cpy`. Pri práci s väčším objemom dát sa tak nemusia tieto dáta zbytočne kopírovať do kontajneru (čím sa vytvára fyzická kópia toho istého súboru na pevnom disku), ale sú len nazdieľané tak, že s nimi môže kontajner pracovať.

3.6 Ovládanie programu pomocou skriptu fixit

Na zjednodušnie ovládania programu bolo vytvorené používateľské rozhranie v podobe Shell skriptu `fixit`, nesúceho meno podľa mena projektu v platforme Testos. Úlohou tohto skriptu je spušťať jednotlivé kroky programového riešenia a navyše uľahčiť ovládanie súčastí ako SSH, alebo VNC pomocou parametrov zadaných do príkazového riadku. Tieto parametre sú:

- `config` - Spustí sa skript `template.py` s parametrom `--no-dialogue`, čím sa vykoná konfigurácia bez dialógu.
- `dialogue` - Spustí sa skript `template.py` bez parametrov, čím sa vyvolá interaktívny dialóg.
- `build` - Spustí sa príkaz `docker build` s parametrami danými konfiguráciou
- `launch` - Spustí sa príkaz `docker run` s parametrami danými konfiguráciou

- `ssh` - Pripojenie pomocou klienta SSH na užívateľa daného konfiguráciou.
- `vnc` - Pripojenie na server VNC kontajneru.
- `clean-all` - Po spustení s týmto argumentom sa vymaže vytvorený kontajner, obraz a všetky ostatné súbory vytvorené pri konfigurácii.
- `remove-image` - Vymaže obraz z lokálneho repozitára Dockeru.
- `remove-container` - Odstráni kontajner z lokálneho systému.
- `run-all` - Spustí skript `main.sh`, ktorý prevezme všetko riadenie a automatizcky vykoná všetky potrebné úkony.
- `-h, help` - Zobrazí nápovedu ku skriptu.

Kapitola 4

Implementačné detaily

Táto kapitola popisuje detaily implementácie jednotlivých skriptov spomenutých v predchádzajúcich kapitolách. Čitateľ má možnosť pochopiť, ako jednotlivé skripty fungujú (využité funkcie, implementované triedy, atď.).

4.1 Zdrojový súbor template.py

Python skript **template.py** implementuje ako hlavnú súčasť triedu `Container`, ktorá zaobaľuje konfiguráciu kontajnera, ktorého predpis sa má za behu skriptu vytvoriť.

4.1.1 Spracovanie konfiguračného súboru config.yml

Spracovanie súboru **config.yml** je zabezpečené funkciou `parse_recipe(container)` skriptu template.py. Funkcia si na začiatku otvorí konfiguračný súbor a jeho obsah načíta pomocou funkcie `yaml.load()` obsiahnutej v module **yaml**. Načítaný obsah je uložený do premennej vo forme slovníka, kde je kľúčom meno položky v konfiguračnom súbore a hodnota obsahuje textovú, číselnú, alebo hodnotu typu boolean.

Funkcia ďalej postupuje jednoduchou validáciou hodnôt a kontroluje, či niektorá položka nechýba, alebo neobsahuje nesprávnu hodnotu. Zároveň dáta ukladá do premenných instancie triedy `Container`, ktorú dostane na vstupe.

4.1.2 Interaktívny dialóg v skripte template.py

Pri spustení skriptu template.py bez argumentov sa spustí funkcia `run_dialogue()`, ktorá má za úlohu používateľa previesť cez konfiguráciu, čím zaniká nutnosť písať súbor config.yml manuálne.

Hlavným prostriedkom `run_dialogue()` je funkcia `input()`, ktorá zo štandardného vstupu načítava textové reťazce zadané používateľom. Tieto údaje sú následne ukladané do slovníka s rovnakou štruktúrou ako slovník popísaný v predchádzajúcej sekcii.

V dialógu sa vyskytujú otázky s odpoveďou áno/nie (yes/no), ktoré sú spracovávané pomocnou funkciou `yes_no(question)`. Táto funkcia na vstup prijíma reťazec `question` obsahujúci otázku, ktorá sa užívateľovi zobrazí na štandardný výstup. Funkcia kontroluje, či užívateľ zadal valídnu odpoveď (yes, alebo no, alternatívne y, alebo n) a v prípade, že tak neučinil, je vyzvaný, aby svoju odpoveď zopakoval.

Neskôr bola doimplementovaná súčasť, ktorá povoľovala užívateľovi niektoré hodnoty nevyplniť s tým, že sa aplikovala implicitná hodnota napísaná pri odpovedi.

4.1.3 Vytváranie súborov pomocou skriptu template.py

Implementačne je vytváranie jednotlivých súborov (sekcia 3.4) rozdelené do samostatných metód triedy `Container`. Každý súbor je tvorený vlastnou metódou a to v tomto poradí:

- Metóda **`Container.print_script_config()`** najprv vytvorí konfiguračný súbor **config**.
- Metóda **`Container.print_host_control_script()`** ďalej vytvorí riadiaci skript **host_control.sh**.
- Metóda **`Container.print_container_control_script()`** vytvorí riadiaci skript **container_control.sh**.
- Metóda **`Container.print_dockerfile()`** nakoniec vytvorí Dockerfile, podľa ktorého sa zostaví Docker obraz.

Všetky potrebné informácie metódy čerpajú z instančných premenných. Vnútri metód je obsah súborov tvorený konkatenáciou reťazcov, ktoré obsahujú fragmenty kódu. Tieto fragmenty sú postupne spájané a podľa prepínačov riadených instančnými premennými sa rozhoduje, či sa daný reťazec do súboru pridá, alebo nie. Tento postup je ovplyvnený požadovanými vlastnosťami cieľového Docker obrazu.

4.2 Shell skripty

Shell skripty slúžia ako už na ovládanie hostiteľského, tak aj na ovládanie kontajnerizovného systému. Aj napriek faktu, že je možné volať systémové príkazy v Python skripte pomocou funkcie `call()` bol zvolený prístup vytvorenia Shell skriptov pomocou template.py. Tento postup bol zvolený kvôli tomu, aby mal používateľ jednoduchý prehľad o funkciách a programoch volaných v rôznych etapách programu.

Vytvorené skripty primárne pozostávajú z volaní programov ako napríklad `sshd` (démon serveru SSH), alebo serveru VNC.

Kapitola 5

Overovanie funkcionality

Táto kapitola popisuje, ako boli testované jednotlivé časti programového riešnia. Čitateľa oboznámi o tesovaní primárných častí a následne ich spojenia do fungujúceho celku, no objasňuje aj prekážky odhalené testami.

5.1 Testovanie skriptu template.py

Najobsiahlejšou súčasťou programového riešenia bol skript **template.py** a preto sa aj väčšina testov zameriavala na neho.

Testovanie tvorby skriptov bolo vykonávané porovnávaním výstupných súborov. Pre potreby testov bolo manuálne vytvorených niekoľko konfiguračných súborov (config.yml) a korešpondujúcich skriptov. Jednotilvé konfiguračné súbory boli postupne načítavané a spracovávané skriptom template.py, ktorý vytváral skripy. Tieto skripty boli následne porovnané so skriptami vytvorenými predom.

Najčastejšími rozdielmi boli veci ako odsadenie riadkov, medzery medzi inštrukciami apod. Podľa týchto výsledkov boli opravené textové reťazce, ktoré sa konkatenovali (sekcia 4.1.3).

Testovanie však odhalilo aj niekoľko chýb týkajúcich sa načítavania a spracovávania súboru config.yml. Zistilo sa, že podmienky ovládajúce niektoré prepínače boli naimplementované nesprávne z čoho vyplývalo, že sa konfiguračný súber neinterpretoval správane a vo výstupných súboroch boli fragmenty kódu naviec, alebo chýbali. Po opravení chýb v kóde prebehli všetky testy v poriadku.

Popri testovaní bola zistená aj nekompatibilita s interpretom Python pred verziou 3. Chyba, ktorá to odhalila sa prejavila po refaktorizácii funkcie implementujúcej interaktívny dialóg. Bola spôsobená rozdielnou implementáciou funkcie `input()` - Zatiaľ čo v Python 3 vracia funkcia `input()` textový reťazec, vo verzii 2 táto funkcia vracia Pythonovský výraz.

5.2 Testovanie celku

Testovanie programu ako celku bolo uskutočňované manuálne. Pre účely týchto testov bol použítý zoznam požiadavok na výsledný produkt. Pre jednotlivé body zoznamu boli vytvorené testovacie prípady, ktoré ich pokrývali. Testovacie prípady testovali napríklad:

- funkčnosť VNC serveru v kontajneri,
- prítomnosť súborov, ktoré sa mali do kontajnera nakopírovať počas stavby obrazu,

- vytvorenie užívateľa podľa špecifikácie testera.

Pre tieto testy boli vytvorené vlastné konfiguračné súbory a tiež aj riadiace skripy (container_control_user.sh). Po dokončení skriptov boli výsledky overené pomocou VNC prehliadačov, pripojenia pomocou SSH, alebo príkazu docker attach.

Vzhľadom na využitie kontajnerov práce nebolo možné úplne korektne automatizovať testovanie ich funkcionality. Celý program sa vo veľkej miere spolieha na softvér tretej strany (balíkové manažéry, inštalované balíky atď.), čo do systému zanášalo istú mieru nespoľahlivosti. Tým pádom sa stávalo (najmä v prípade balíkových manažérov), že sa pri aktualizácii staršieho obrazu zapli neočakávané dialógy, ktorých správanie nebolo možné jednoducho ošetriť a spôsobovalo to napríklad to, že sa stavba obrazu prerušila kvôli chybému hláseniu vyvolanému nesprávnou odpoveďou na dialóg.

Tento prípad bol odhalený, keď pri aktualizácii systému Arch Linux jeden z balíkov vyžadoval pridanie dodatočných bezpečnostných kľúčov. Aj keď bol zadaný prepínač, ktorý mal pri inštalácii balíka užívateľa zbaviť nutnosti tento proces nejako ovplyvňovať, na pridanie kľúča nestačil a inštalácia balíka sa predčasne skončila a vrátila chybový návratový kód, čo spôsobilo, že sa skladanie kontajnera zastavilo.

Ďalším príkladom nedostatku balíkového manažéra bol prípad v systéme Fedora. Pri inštalácii balíkov zrejme nastal výkyv rýchlosti sťahovania balíka a balíkový manažér kvôli tomu daný balík nenainštaloval. Toto síce nespôsobilo chybové hlásenie, no vo výslednom obraze tento balík chýbal a tak nebolo možné vykonať testy.

5.3 Testy založené na požiadavkách

Pre testovanie a následnú demonštráciu boli vytvorené testy, ktoré mali za účel otestovať splnenie jednotlivých požiadavok (sekcia 3.1).

1. **test_true** - Tento test má za úlohu otestovať základnú funkcionalitu programu (požiadavka 6). Tvorí ho základný kontajner bez dodatočnej konfigurácie, v ktorom sa vytvorí súbor obsahujúci reťazec "true"s menom log.txt. Test je úspešný, pokiaľ sa k súboru používateľ dostane cez zdieľaný priečinok z hostiteľského systému.

2. **test_copy_files** - Úlohou testu je pri tvorbe obrazu do neho nakopírovať súbor (požiadavky 4 a 6). Tento súbor je potom nakopírovaný do zdieľaného priečinka. Test je úspešný, pokiaľ sa k súboru používateľ dostane cez zdieľaný priečinok z hostiteľského systému.

3. **test_vnc** - Tento test má za úlohu otestovať funkcionalitu kontajneru rozšírenú o server VNC (požiadavky 1, 3, 3.a). Program spustí v kontajneri server VNC a program firefox. Užívateľ sa potom môže na kontajner pripojiť a interagovať s programom pomocou jeho grafického používateľského rozhrania.

4. **test_screenshot** - Tento test vychádza z testu **test_vnc** a rozširuje ho o funkcionalitu, kde sa po 10 sekundách vytvorí snímok obrazovky vzdialenej plochy (požiadavka 3.c). Tento snímok je po dokončení dostupný v zdieľanom priečinku.

5. **test_ssh** - Test má za úlohu spustiť kontajner s aktívnym SSH serverom (požiadavka 2). Po jeho spustení je užívateľ schopný pripojiť sa na kontajner pomocou klienta SSH, alebo za využitia dodaného používateľského rozhrania **fixit.sh**.

6. **test_all** - Tento test demonštruje splnenie všetkých požiadavok a teda obsahuje spustený server SSH, server VNC so správcom okien, spúšťa program firefox a kopíruje súbory.

Funkcionalita programu je podložená využitím zdieľaného média a tak požiadavku 6 spĺňajú všetky testy.

Všetky zmienené testy sú k tejto práci priložené na DVD (príloha A) v adresári `/tests`. Každý test obsahuje vlastné zdrojové súbory s konfiguráciou a popis, ako ho spustiť a čo má byť výsledkom. Toto DVD tiež nesie obraz virtuálneho systému Lubuntu, ktoré obsahuje testy (adresár `/home/fixit/fixit/`) a všetky potrebné závislosti.

Kapitola 6

Záver

Cieľom práce bolo naštudovať platformu Docker a následne naimplementovať program, ktorý by ju zaobaľoval a umožňoval prácu s ňou užívateľovi, ktorý s ňou nie je oboznámený.

Pomocou skriptu napísanom v jazyku Python sa spracuje konfigurácia a vytvoria sa riadiace Shell skripty a Dockerfile, podľa ktorého sa poskladá obraz cieľového systému. Súčasťou následnej stavby je inštalácia balíkov vyžadovaných užívateľom, konfigurácia súčastí, ako napríklad server SSH, alebo server VNC, odhalenie sieťových portov kontajnerizovaného systému a skopírovanie súborov do kontajneru. Nakoniec riadenie preberá Shell skript, ktorý zavolá stavbu obrazu a spustí podľa neho kontajner, v ktorom sa vykoná záverečné nastavenie systému a samotné testovanie. Porovnaním počiatočných požiadaviek na výsledný produkt sa dá tvrdiť, že splňuje všetky, avšak nie za každých okolností (sekcia 5.2).

Hlavný prínos práce sa ukázal pri testovaní a používaní, kedy bol užívateľ oboznámený s ovládaním schopný kontajner vytvoriť a nasadiť v priebehu niekoľkých momentov bez znalosti platformy Docker, čo ukazuje, že projekt spĺňa svoj účel podľa predpokladov. Za využitia technológie VNC je tiež možné v kontajneri testovať grafické aplikácie, čím program spĺňa svoj primárny účel.

Čo sa týka budúcnosti projektu, program bude ďalej vyvíjaný pod projektom Testos. Medzi plánované súčasti patrí napríklad rozšírené používateľské prostredie ovládajúce ako už pôvodnú tak aj novú funkcionalitu, jeho grafická nadstavba, vytvorenie multikontajnerového prostredia (napríklad vo vzťahu klient a server), nastavenie sieťových podmienok kontajneru, alebo možnosť zabalenia aplikácie z hostiteľského systému s jej všetkými závislosťami a nastaveniami do jej vlastného kontajneru, v ktorom sa má testovať.

Literatúra

[1] Docker, Inc.: *Dockerfile Reference*. 2017, [Online; navštívené 24.4.2017].
URL https://docs.docker.com/engine/reference/builder/

[2] FINK, John: Docker: a software as a service, operating system-level virtualization framework. *Code4Lib Journal*, ročník 25, 2014, ISSN 1940-5758, [Online; navštívené 20.3.2017].
URL http://journal.code4lib.org/articles/9669

[3] JANOUŠEK, Martin: Dynamické analyzátory pro platformu SearchBestie [online]. 2017.
URL https://pajda.fit.vutbr.cz/jct/searchBestie

[4] MALÍK, Viktor: *Dynamická analýza použití knihovních volání*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2014.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=15992

[5] MESZAROS, Gerard: *xUnit test patterns: Refactoring test code*. NJ: Addison-Wesley, 2007, ISBN 9780131495050.

[6] Oracle: *Oracle Solaris Zones*. 2017, [Online; navštívené 19.03.2017].
URL https://docs.oracle.com/cd/E18440_01/doc.111/e18415/chapter_zones.htm#OPCUG426

[7] Red Hat, Inc.: *Introduction to Control Groups*. 2017, [Online; navštívené 12.03.2017].
URL https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html

[8] RICHARDON, Tristan; WOOD, Kenneth R: The RFB protocol. *ORL, Cambridge, January*, 1998.

[9] RICHARDSON, Tristan, et al: Virtual network computing. *IEEE Internet Computing*, ročník 2, č. 1, 1998: s. 33–38, ISSN 1089-7801.

[10] ROSEN, Rami: *Linux Containers and the Future Cloud*. Linux Journal, 2014.
URL http://www.linuxjournal.com/content/linux-containers-and-future-cloud

[11] Skupina Testos: Domovská stránka projektu Testos [online]. FIT VUT v Brně, 2017.
URL http://testos.org

[12] The FreeBSD Project: *Chapter 14. Jails*. 2016, [Online; navštívené 19.03.2017].
URL https://www.freebsd.org/doc/handbook/jails.html

[13] WALSH, Daniel J.: *Secure Linux Containers*. 2013, [Online; navštívené 12.03.2017].
URL https://www.usenix.org/conference/lisa13/secure-linux-containers

[14] YLONEN,Tatu; LONVICK Chris: *The Secure Shell (SSH) Transport Layer Protocol*. 2006, [Online; navštívené 24.3.2017].
URL https://tools.ietf.org/html/rfc4253

Prílohy

Príloha A

Obsah priloženého pamäťového média

```
/
├── src/ ... zdrojové súbory programového riešenia
├── tex_src/ ... zdrojové súbory technickej správy vo formáte LaTeX
├── tests/ ... testovacia sada určená na overenie požiadavkov a
│             demonštráciu funkcionality
│   ├── README.md ... readme súbor opisujúci, ako pracovať s testami
│   └── test_*/ ... adresáre obsahujúce jednotlivé spustiteľné testy a ich
│                   popis
├── README.md ... súbor obsahujúci popis jednotlivých adresárov a súborov a
│                 návod ako spustiť program
├── fixit.ova ... obraz virtuálneho stroja obsahujúci zdrojové súbory a testy
│                 a potrebné závislosti
└── xorsza00.pdf ... technická správa exportovaná do formátu pdf
```